BAB III

ANALISA DAN PERANCANGAN SISTEM

3.1 Analisa Masalah

Analisa masalah yang didapat dari penelitian ini adalah memperbaiki algoritma RC4 yang dirasa kurang aman saat diimplementasikan untuk pengamanan file. Performa algoritma RC4 dapat dianalisa melalui waktu yang dibutuhkan untuk melakukan pembangkitan kunci, enkripsi, dekripsi. Hasil pengujian performa algoritma RC4 standar dijadikan bahan pertimbangan untuk memodifikasi algoritma tersebut. Acuan modifikasi algoritma RC4 berasal dari penelitian yang dicetus oleh Sura M. Searan dan Ali M. Sagheer [5] namun berbeda pada media yang dipakai.

Kelemahan algoritma RC4 pada fase KSA menjadi celah keamanan yang memungkinkan terjadinya berbagai jenis serangan. Jenis-jenis metode serangan ini telah dijelaskan pada pada sub bab 2.2.4 . Tujuan utama penggunaan algoritma RC4 adalah untuk keamanan pengguna. Celah keamanan pada algoritma RC4 harus segera dibenahi selain memperbaiki performa algoritma tersebut. Hasil modifikasi algoritma RC4 diharapkan dapat membenahi celah keamanan pada algoritma RC4. Untuk membuktikan hal ini, pengujian keamanan dilakukan. Pengujian keamanan membandingkan algoritma RC4 standart dan algoritma RC4 modifikasi digunakan untuk menganalisa keamanan antara kedua algoritma tersebut. Metode uji keamanan yang digunakan antara *brute force attack* dan *bit flipping attack.* Hasil pengujian performa dan pengujian keamanan diharapkan dapat menjadi bahan evaluasi untuk mengukur kualitas hasil modifikasi algoritma RC4 baik dari segi performa maupun keamanan.

3.2 Rancangan Algoritma RC4 Standar

Rancangan algoritma RC4 standar ini sesuai dengan algoritma RC4 asli pada sub bab 2.2. Rancangan ini nantinya diimplementasikan pada program komputer yang ditulis dengan Bahasa pemrograman java. Algoritma ini terbagi menjadi dua mekanisme yang berbeda yang dilakukan secara berurutan.

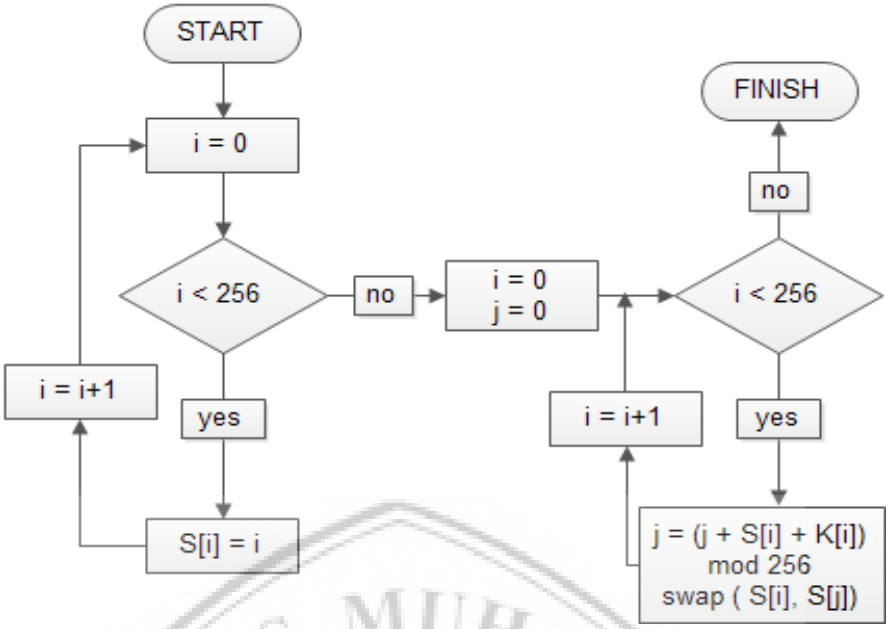
3.2.1 Fase KSA (*Key Scheduling Algorithm*) pada RC4 Standar

Key Scheduling Algorithm (KSA) merupakan tahapan pemberian nilai awal berdasarkan kunci enkripsi. State dari nilai awal tersebut berupa array dengan representasi permutasi 256 byte (dengan indeks o sampai 255) dinamakan array S. Proses inisialisasi S-Box (Array S) berisi S[0], S[1],….., S[255], dengan bilangan 0 sampai 255. Pengacakan S-Box dimulai dengan iterasi j = 0 dengan mengulangi penghitungan menggunakan rumus 1.0. Pada proses ini juga dilakukan inisialisasi S-Box untuk Array Kunci/ Panjng kunci K=[i mod panjang kunci]. Pengacakan dilakukan dengan menukar isi S[i] dan S[j], sehingga menghasilkan output state. Pseudo code proses penjadwalan kunci algoritma RC4 standar adalah sebagaimana gambar 3.1 :

```
Input   : Kunci
Output  : State
1. For i = 0 to 255 do
    1.1  State[i] = i
2. Set j = 0
3. For ( i = 0 to 255) do
    3.1  j = ( j + State[i] + Kunci [ i mod panjang kunci] ) mod
         256  . . . . . . . . . . . . . . . . . . . . (rumus 1.0)
    3.2  Swap ( State[i], State[j]) . . . . . . . . (rumus 1.1)
Output : State
```

Gambar 3.1: Pseudocode Penjadwalan Kunci pada Algoritma RC4 Standar

Inputan berawal dari inisialisasi i= 0 apabila i < 256 maka dilakukan penambahan i = i + 1 hingga memenuhi array S-Box, jika hasil tidak memenuhi i<256 maka dilakukan pengacakan S-Box. Pengacakan S-Box dimulai dengan iterasi j = 0 dengan mengulangi penghitungan menggunakan rumus 1.0 diikuti dengan penukaran pada rumus 1.1. Pengacakan ini diulangi hingga i < 256. Alur proses penjadwalan kunci pada RC4 standar adalah sebagai berikut :



Gambar 3.2 : Flowchart Penjadwalan Kunci pada Algoritma RC4 Standart

3.2.2 Fase PRGA (*Pseudo-Random Generaton Algorithm*) pada RC4 Standart

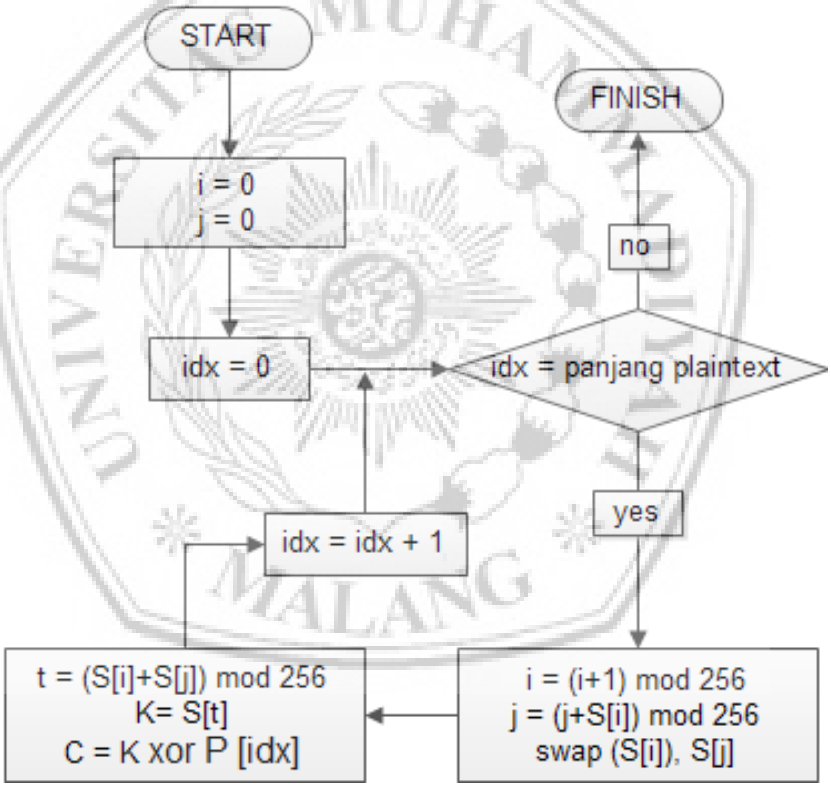
Tahap PRGA terjadi modifikasi state dan output sebuah byte dari aliran kunci, dimana array S beroprasi dengan array U yang selanjutnya akan menghasilkan keystream. Inisialisai dimulai dengan i = 0 dan j = 0, i = 0 diulangi hingga memenuhi plaintext dengan penambahan pada rumus 2.0 pada awal proses penghitungan iterasi dengan rumus 2.1. Kemudian ditukar antara State[i] dan State[j] menggunakan rumus 2.2. Proses mendapatkan Key Sequence dengan perhitungan pada rumus 2.3. Outputan Key di XOR kan dengan plaintext untuk menghasilkan chipertext. Pseudo code proses PGRA algoritma RC4 standar adalah sebagai berikut :

```
 Input: State, Plaintext
Output: Key Sequence (Stream teks sandi)
1. Intialisasi :
     1.1  i = 0
     1.2  j = 0
2. For ( i = 0 to Plaintext )
     2.1  i = ( i +1 ) mod 256 . . . . . .(rumus 2.0)
     2.2  j = ( j + State[i]) mod 256  . . . . . .(rumus 2.1)
     2.3  Swap (State[i], State[j]). . . . . .(rumus 2.2)
     2.4  Key Sequence = State (State[i] +
          State[j]) mod 256 . . . . . (rumus 2.3)
```

```
3. Output : Key Sequence
4. Keluaran Key Sequence di XOR dengan Plaintext
    : Ci= Ki ⊕ Plaintext. . . . . .(rumus 2.4)
```

Gambar 3.3 : Pseudocode Proses Enkripsi pada RC4 Standar

Inputan inisialisasi dimulai dengan i = 0 dan j = 0. Index = 0 melakukan proses proses penghitungan dengan rumus 2.0 dan 2.1. Kemudian ditukar antara State[i] dan State[j] seperti pada rumus 2.2. Key Sequence didapatkan setelah menghitung dengan rumus 2.3. Index diulang sampai memenuhi panjang plaintext dengan menambahkan idx = idx+1. Hasil outputan key tersebut di-XOR-kan dengan plaintext menggunakan rumus 2.4 untuk menghasilkan chipertext. Alur proses PGRA algoritma RC4 standar adalah sebagai berikut :



Gambar 3.4 : Flowchart Proses Enkripsi RC4 Standar

3.3 Rancangan Modifikasi Algoritma RC4

Rancangan modifikasi algoritma RC4 ini sesuai dengan jurnal milik Sura M. Searan dan Ali M Sagheer yang telah dijelaskan pada sub bab 2.3. Rancangan algoritma ini mempunyai dua initial state dan salah satu state bernilai state factorial pada fase KSA yang hal ini membedakan dari algoritma RC4 standar.

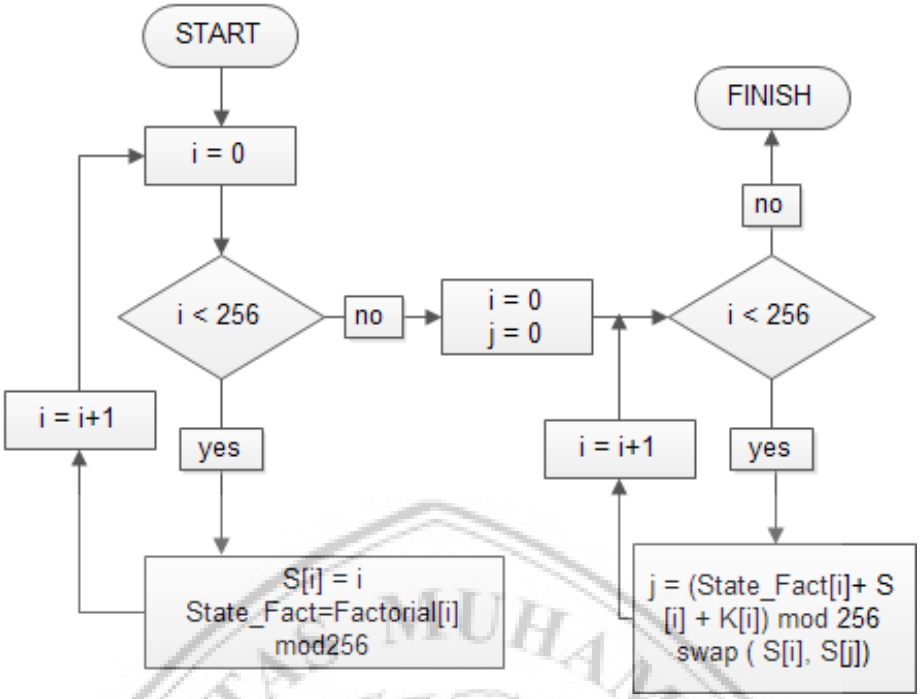
3.3.1 Fase KSA (*Key Scheduling Algorithm*) pada Modifikasi RC4

Proses inisialisasi S-Box (Array S) berisi S[0], S[1],….., S[255], dengan bilangan 0 sampai 255. State factorial berisi nilai state S_Fact[0]= Faktorial 0 mod 256 sampai memenuhi array S_Fact[i] pada rumus 3.0. Pengacakan S-Box dimulai dengan iterasi j = 0 dengan mengulangi penghitungan pada rumus 3.1. Pada proses ini juga dilakukan inisialisasi S-Box untuk Array Kunci/ Panjng kunci K=[i mod panjang kunci]. Pengacakan dilakukan dengan menukar isi S[i] dan S[j] menggunakan rumus 3.2 , sehingga mengasilkan output state.

```
Input: Key
Output: State [ i ]
1. For i = 0 to 255 do
   State[ i ] = i
2. State_Fact [ i ]= Factorial ( i ) mod 256 . . . . . (rumus 3.0)
3. Set j = 0
4. For ( i = 0 to 255) do
   4.1  j = ( State_Fact [ i ] +  State[ i ] + Key [ I mod [ Key ]
            ] ) mod 256 . . . . . (rumus 3.1)
   4.2  Swap ( State [ i ], State [ j ] ) . . . . . (rumus 3.2)
5. Output : State [ i ]
```

Gambar 3.5 : Pseudocode Penjadwalan Kunci pada Modifikasi Algoritma RC4

Inputan diawali dengan inisialisasi i= 0 apabila i < 256 dengan dua buah kotak subsitusi (S-Box) dan state factorial pada rumus 3.0 maka dilakukan penambahan i = i + 1 hingga memenuhi array S-Box, jika hasil tidak memenuhi i<256 maka dilakukan pengacakan S-Box. Pengacakan S-Box dimulai dengan iterasi j = 0 dengan mengulangi penghitungan pada rumus 3.1 diikuti dengan penukaran S[i] dan S[j] dengan menggunakan rumus 3.2. Pengacakan ini diulangi hingga i < 256.



Gambar 3.6 : Flowchart Penjadwalan Kunci pada Modifikasi Algoritma RC4

3.3.2 Fase PRGA (*Pseudo-Random Generaton Algorithm*) pada Modifikasi RC4

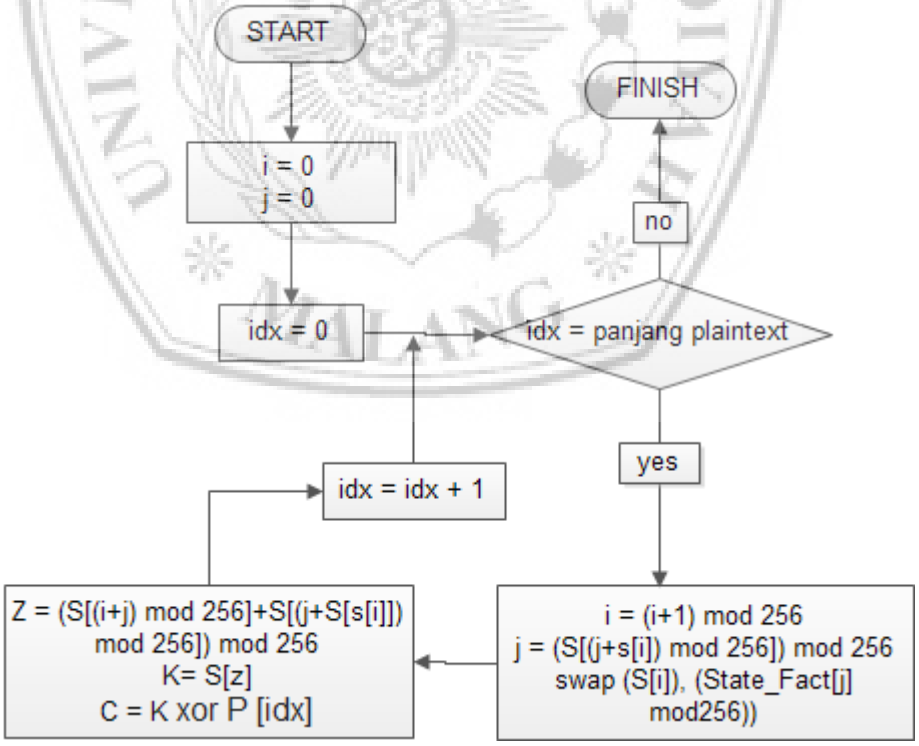
Proses PRGA pada modifikasi algoritma RC4 diawali dengan menginisialisasi i = 0 dan j = 0 i = 0 diulangi hingga memenuhi plaintext seperti rumus 4.0 pada pada awal proses penghitungan iterasi j menggunakan rumus 4.1. Kemudian ditukar antara State pada rumus 4.2. Proses mendapatkan Key Sequence dengan menghitung K pada rumus 4.3. Outputan key di XOR dengan plaintext menggunakan rumus 4.4 untuk menghasilkan chipertext.

```
Input  : State [ i ], Plaintext
Output : Key Sequence (Stream teks sandi)
1.  Set Intialisasi i = 0, j = 0
2.  Output Generation loop
    2.1  i = ( i +1 ) mod 256 . . . . . (rumus 4.0)
    2.2  j = ( State [( j + State [ i ] mod 256])mod 256 . . . . .
                    . . . . . . . . . . . . . . . . . (rumus 4.1)
    2.3  Swap (State[ i ], (State_Fact [ j ] mod 256)). (rumus 4.2)
    2.4  Z = ( State [( i + j ) mod 256] + State
         [( j + State[state [ i ]] mod 256 ]) mod 256 . (rumus 4.3)
    2.5  Key Sequence = State Z
    2.6  Output : Key Sequence
```

```
2.  Keluaran Key Sequence di XOR dengan  Plaintext:
    Ci= Key Sec ⊕ Plaintext. . . . . . . . . . . . . . . (rumus 4.4)
```

Gambar 3.7 : Pseudocode Enkripsi pada Modifikasi Algoritma RC4

Inputan inisialisasi dimulai dengan i = 0 dan j = 0. Index = 0 melakukan proses penghitungan dengan rumus 4.0 dan 4.1. Kemudian menukar antar state dengan rumus 4.3. Key Sequence didapatkan dengan menghitung memakai rumus 4.4. Index diulang sampai memenuhi panjang plaintext dengan menambahkan idx = idx+1. Hasil outputan key tersebut di-XOR-kan dengan plaintext menggunakan sehingga menghasilkan chipertext. Penelitian kali ini, sistem yang akan dirancang berbasis desktop dengan menggunakan bahasa pemrograman java. Inputan algoritma ini adalah awal state berupa nilai 1 sampai 255 dan kunci rahasia dengan panjang antara 1 sampai dengan 256, dan *state table* dengan 256 byte mengandung factorial dari keadaan nilai state awal. Pelaksanaan yang diusulkan algoritma ini diperlukan waktu kurang dari yang dibutuhkan untuk pelaksanaan RC4 standart. Ketika diimplementasikan pada ukuran yang sama dari kunci rahasia.



Gambar 3.8 : Flowchart Enkripsi pada Modifikasi Algoritma RC4

3.4 Rancangan Uji Keamanan

Algoritma RC4 memiliki banyak kerentanan yang dapat dipergunakan untuk memecahkan keamananya. Informasi yang didapatkan setelah berhasil melakukan serangan antara lain menemukan, mengubah atau merusak file asli. Rancangan pengujian ini dipergunakan untuk menganalisis perbandingan tingkat keamanan antara RC4 standar dan RC4 yang telah dimodifikasi.

3.4.1 Brute force Attack

Brute force attack merupakan metode penyerangan yang digunakan untuk mendapatkan semua kemungkinan hasil dari *plaint text* dengan menggunakan seluruh percobaan terhadap semua kunci. Teknik ini adalah teknik yang paling banyak digunakan untuk memecahkan password, kunci, kode atau kombinasi. Cara kerja metode ini sangat sederhana yaitu mencoba semua kombinasi yang mungkin.

Penggunaan *brute force* attack salah satunya adalah *password cracker*. Sebuah password dapat dibongkar dengan menggunakan program bernama password cracker ini. Program password cracker adalah program yang mencoba membuka sebuah password yang telah terenkripsi dengan menggunakan sebuah algoritma tertentu dengan cara mencoba semua kemungkinan. Teknik ini sangatlah sederhana, tapi efektivitasnya luar biasa, dan tidak ada satu pun sistem yang aman dari serangan ini, meski teknik ini memakan waktu yang sangat lama, khususnya untuk password yang rumit.

3.4.2 Bit Flipping Attack

Bit flip attack merupakan serangan pada *stream cipher* dengan tujuan untuk mengubah hasil dekripsi dengan cara mengubah bit *ciphertext* tertentu. Proses pengubahan tersebut dilakukan dengan melakukan proses flip (membalikkan) bit tertentu pada *ciphertext*. Maksud dari proses flip tersebut adalah mengubah 0 jadi 1 atau 1 jadi 0.

Kelemahan RC4 yang paling dikenal adalah *Bit Flipping Attack* yang dapat menjadi sangat berbahaya apabila seseorang mengganti sebagian informasi dari *plaintext*. Walaupun punya banyak kelemahan, tetapi metode enkripsi RC4 saat ini masih menjadi metode enkripsi yang banyak digunakan.

Contoh kasus:

Pesan plaintext

oke = 011011110110101101100101

Keystream

123 = 001100010011001000110011

Keystream di-XOR-kan dengan plaintext menghasilkan ciphertext :

011011110110101101100101

001100010011001000110011 xor

010111100101100101010110

Attacker melakukan flip:

000111100101100101010110000

Ciphertext di dekripsi dan didapatkan plaintext

001011110110101101100101 , maka pesan sudah mengalami kerusakan.

3.5 Perbedaan penghitungan algoritma RC4 standar dan modifikasi RC4

Mengarah ke pembahasan perbandingan dari algoritma RC4 standar dan modifikasi algoritma RC4 itu bekerja, perlu dibahas dahulu pembuktian bagaimana cara kerja kedua algoritma berikut. Contoh penerapan enkripsi dan dekripsi pada algoritma RC4 standart dengan *state-array* 4 *byte*, kenapa state-array 4 *byte*, jika kita menyekesaikan secara manual dengan state-array 256 byte ini akan banyak memakan waktu lama dalam menyelesaikan satu persatu inisialisasi array S-Box. Dalam penerapan ini kita akan mengenkripsi kata "**sony"** dengan kunci dengan kunci "**abcd"**.

Contoh fase KSA pada algoritma RC4 standar adalah berikut :

Pertama inisialisasi S-Box dengan panjang kunci 4 *byte*, dengan S[0]=0, S[1]=1, S[2]=2, S[3]=3 dengan panjang kunci 4 byte juga, sehingga terbentuk Array S dan Arra K seperti tabel dibawah :

Array S	0	1	2	3
Array K	a=97	b=98	c=99	d=100

Membuat table state-array, Berikutnya mencampur operasi dimana kita akan menggunakan variable i dan j ke indeks S[i] dan K[j]. Pertama kita beri nilai inisial untuk i dan j dengan 0. Operasi pencampuran adalah pengulangan rumusan

(j + S[i] + K[i]) mod 4 yang diikuti dengan penukaran S[i] dengan S[j]. Karena menggunakan *array* dengan panjang 4 byte maka algoritmanya menjadi :

For i = 0 to 4

J = (j + S[i] + K[i]) mod 4

Swap S[i] dan S[j]

Dengan algoritma seperti diatas maka nilai awal i = 0, sampai i = 3 akan menghasilkan *array* S seperti berikut :

Iterasi pertama :

i = 0, maka

j = (j + S[i] + K[i]) mod 4

= (j + S[0] + K[0]) mod 4

= (0 + 0 + 97) mod 4

= 1

Swap S[0] dan S[1] sehingga menghasilkan:

Array S	1	0	2	3

Iterasi kedua :

i = 1, maka

j = (j + S[i] + K[i]) mod 4

= (j + S[1] + K[1]) mod 4

= (1 + 0 + 98) mod 4

= 3

Swap S[1] dan S[3] sehingga menghasilkan:

Array S	1	3	2	0

Iterasi ketiga :

i = 2, maka

j = (j + S[i] + K[i]) mod 4

= (j + S[2] + K[0]) mod 4

= (3 + 2 + 99) mod 4

= 0

Swap S[2] dan S[0] sehingga menghasilkan:

Array S	2	3	1	0

Iterasi keempat :

i = 3, maka

j = (j + S[i] + K[i]) mod 4

= (j + S[3] + K[0]) mod 4

= (0 + 0 + 100) mod 4

= 0

Swap S[3] dan S[0] sehingga menghasilkan:

Array S	0	3	1	2

Contoh fase KSA pada modifikasi algoritma RC4 dengan menggunakan soal yang sama adalah sebagai berikut:

Pertama inisialisasi S-Box dengan panjang kunci 4 *byte*, dengan S[0]=0, S[1]=1, S[2]=2, S[3]=3 dengan panjang kunci 4 *byte* dan untuk *State factorial* berisi nilai *state* S_Facct[0]=Factorail[i] mod 4, sehingga terbentuk *Array* S, *Array* S_Fact dan *Array* K seperti pada table dibawah:

Array S	0	1	2	3
Array S_Fact	1	1	2	2
Array K	a=97	b=98	c=99	d=100

Membuat table state-array, berikutnya mencampur operasi dimana kita akan menggunakan variable i dan j ke indeks S[i] dan S[j]. Pertama kita beri nilai inisial untuk i dan j dengan 0. Operasi pencampuran adalah pengulangan rumusan (S_Fact[i]+S[i]+K[i]) mod 4 yang diikuti dengan penukaran S[i] dengan S[j]. Karena menggunakan *array* dengan panjang 4 byte maka algoritmanya menjadi :
For i = 0 sampai 4

j = (S_Fact[i]+S[i]+K[i]) mod 4

Swap S[i], S[j]

Algoritma diatas maka nilai awal i=0 sampai i=3 akan menghasilkan *array* S seperti berikut:

Iterasi pertama:

i = 0, maka

j = (S_Fact[i] + S[i] + K[i]) mod 4

= (S_Fact[0] + S[0] + K[0]) mod 4

= (1 + 0 + 97) mod 4

= 2

Swap S[0] dan S[2] sehingga menghasilkan:

Array S	2	1	0	3

Iterasi kedua:

i = 1, maka

j = (S_Fact[i] + S[i] + K[i]) mod 4

= (S_Fact[1] + S[1] + K[1]) mod 4

= (1 + 1 + 98) mod 4

= 0

Swap S[1] dan S[0] sehingga menghasilkan:

Array S	1	2	0	3

Iterasi ketiga:

i = 2, maka

j = (S_Fact[i] + S[i] + K[i]) mod 4

= (S_Fact[2] + S[2] + K[2]) mod 4

= (2 + 0 + 99) mod 4

= 1

Swap S[2] dan S[1] sehingga menghasilkan:

Array S	1	0	2	3

Iterasi keempat:

i = 3, maka

j = (S_Fact[i] + S[i] + K[i]) mod 4

= (S_Fact[3] + S[3] + K[3]) mod 4

= (2 + 3 + 100) mod 4

= 1

Swap S[3] dan S[1] sehingga menghasilkan:

Array S	1	3	2	0

Berikut fase PRGA pada algoritma RC4 standar :

Hasil *array* S dari iterai keempat, maka proses selanjutnya yaitu meng-XOR kan kata **sony** sebanyak 4 kali dikarenakan *plaintext* yang akan dienkripsi

berjumlah 4 karakter. Membutuhkan 4 iterasi karena 1 lunci dan 1 kali pengoperasian XOR untuk tiap-tiap karakter pada *plaintext. Array* yang digunakan untuk meng-XOR-kan adalah *array* dari hasil pencarian nilai array terakhir.

Array S	0	3	1	2

Inisialisasi

i = 0

j = 0

Iterasi pertama:

i = (0 + 1) mod 4 = 1

j = (0 + S[1]) mod 4

= (0 + 3) mod 4 = 3

Swap (S[1], S[3])

0	2	1	3

K1 = S [(S [1] + S[3]) mod 4] = S[5 mod 4] = 1

K1 = 00000001

Iterasi kedua:

i = (1 + 1) mod 4 = 2

j = (3 + S[2]) mod 4

= (3 + 1) mod 4 = 0

Swap (S[2], S[0])

1	2	0	3

K2 = S [(S [2] + S[0]) mod 4] = S[1 mod 4] = 3

K2 = 00000011

Iterasi ketiga:

i = (2 + 1) mod 4 = 3

j = (0 + S[3]) mod 4

= (0 + 3) mod 4 = 3

Swap (S[3], S[3])

1	2	0	3

K3 = S [(S [3] + S[3]) mod 4] = S[6 mod 4] = 2

K3 = 00000010

Iterasi keempat

i = (3 + 1) mod 4 = 0

j = (0 + S[0]) mod 4

= (3 + 1) mod 4 = 0

Swap (S[0], S[0])

1	2	0	3

K4 = S [(S [0] + S[0]) mod 4] = S[2 mod 4] = 2

K4 = 00000010

Tabel 3.1 : Hasil Penghitungan Nilai K setiap Iterasi RC4 Standar

K1	K2	K3	K4
00000001	**00000011**	**00000010**	**00000010**

Contoh fase PRGA pada modifikasi algoritma RC4 dengan menggunakan soal yang sama adalah sebagai berikut:

Hasil *array* S dari iterai keempat, maka proses selanjutnya yaitu meng-XOR kan kata **File** sebanyak 4 kali dikarenakan *plaintext* yang akan dienkripsi berjumlah 4 karakter. Membutuhkan 4 iterasi karena 1 lunci dan 1 kali pengoperasian XOR untuk tiap-tiap karakter pada *plaintext. Array* yang digunakan untuk meng-XOR-kan adalah *array* dari hasil pencarian nilai array terakhir.

Array S	1	3	2	0

Inisialisasi

i = 0

j = 0

Iterasi Pertama:

i = (0+1) mod 4 = 1

j = S[(j+S [i]) mod 4]) mod 4

= S[(0+3)mod4] mod 4 = 0

Swap (S[i], (S_Fact[j] mod 4))

Array S	1	1	2	0
Array S_Fact	3	1	2	2

K1 = State [(i+j) mod 4] + State [j + Stat [state [i]] mod 4]) mod 4

K1 = (S [(1+0) mod 4] + S [(0 + S[S [0]]) mod 4]) mod 4 = 2

K1 = 00000010

Iterasi Kedua:

i = (1+1) mod 4 = 2

j = S[(j+S [i]) mod 4]) mod 4

= S[(0+2)mod4] mod 4 = 2

Swap (S[i], (S_Fact[j] mod 4))

Array S	1	1	2	0
Array S_Fact	3	1	2	2

K2 = State [(i+j) mod 4] + State [j + Stat [state [i]] mod 4]) mod 4

K2 = (S [(2+2) mod 4] + S [(2 + S[S [2]]) mod 4]) mod 4 = 2

K2 = 00000010

Iterasi Ketiga

i = (2+1) mod 4 = 3

j = S[(j+S [i]) mod 4]) mod 4

= S[(2+0)mod4] mod 4 = 2

Array S	1	1	2	2
Array S_Fact	3	1	0	2

Swap (S[i], (S_Fact[j] mod 4))

K3 = State [(i+j) mod 4] + State [j + Stat [state [i]] mod 4]) mod 4

K3 = (S [(3+2) mod 4] + S [(2 + S[S [3]]) mod 4]) mod 4 = 2

K3 = 00000010

Iterasi Keempat :

i = (3+1) mod 4 = 0

j = S[(j+S [i]) mod 4]) mod 4

= S[(2+1)mod4] mod 4 = 2

Swap (S[i], (S_Fact[j] mod 4))

Array S	0	1	2	2
Array S_Fact	3	1	1	2

K4 = State [(i+j) mod 4] + State [j + Stat [state [i]] mod 4]) mod 4

K4 = (S [(0+2) mod 4] + S [(2 + S[S [0]]) mod 4]) mod 4 = 0

K4 = 00000000

Tabel 3.2 : Hasil Penghitungan Nilai K setiap Iterasi pada Modifikasi RC4

K1	K2	K3	K4
00000010	00000010	00000010	00000000

Kunci yang ditemukan untuk setiap karakter pada algoritma RC4 standart dan modifikasi RC4, maka dilakukan operasi XOR antar karakter pada plaintext dengan kunci yang dihasilkan. Berikut adalah tabel ASCII untuk tiap karakter pada plaintext yang digunakan.

Tabel 3.3 : Nilai dari kode ASCII

HURUF	KODE ASCII (8 Bit)
s	01110011
o	01101111
n	01101110
y	01111001

Hasil dan proses XOR dengan kunci yang dihasilkan oleh algoritma RC4 standart dapat dilihat pada tabel dibawah ini.

Tabel 3.4 : Hasil Chipertext pada Algoritma RC4 Standar

	s	o	n	y
Plaintext	01110011	01101111	01101110	01111001
Key	00000001	00000011	00000010	00000010
Chipertext	01110010	01101100	01101100	01111011
	r	l	l	{

Hasil dan proses XOR dengan kunci yang dihasilkan oleh modifikasi algoritma RC4 dapat dilihat sebagai berikut :

Tabel 3.5 : Hasil Chipertext pada Modifikasi Algoritma RC4

	s	o	n	y
Plaintext	01110011	01101111	01101110	01111001
Key	00000010	00000010	00000010	00000000
Chipertext	01110001	01101101	01101100	01111001
	q	m	l	y

Hasil enkripsi telah didapatkan maka proses selanjutnya adalah proses pendekripsian, proses ini dimana untuk mengetahui text asli (plaintext) atau mengembalikan nilai sebenarnya, maka dilakukan proses XOR antara kunci dekripsi dan kunci (angka biner) dari chipertext, dimana kunci dekripsi sama dengan kunci enkripsi. Ulasan diatas dapat dijadikan sebagai bahan mencari perbandingan pada kedua algoritma tersebut hingga dari contoh mekanisme kerja algoritma diatas dapat diambil beberapa kesimpulan perbedaan antara algoritma RC4 standar dan modifikasi algoritma RC4 sebagai berikut:

Tabel 3.6 : Perbandingan Mekanisme Kerja Algoritma RC4 Standar dan Modifikasi Algoritma RC4.

Proses	Perbedaan	RC4 Standart	Modifikasi RC4
KSA/ Penjadwalan Kunci	Inisialisasi State	Menggunakan satu State	State_Fact[i]=Factorial(i) mod 256
	Perhitungan nilai j tiap iterasi	j = (j +State[i]+Key[i mod key-length]) mod 256	j = (State_Fact[i]+State[i]+Key[I mod key-length]) mod 256
PRGA	Perhitungan nilai j tiap iterasi	j = (j+State[i]) mod 256	j = (State[(j+state[i]mod 256]) mod 256
	Penukaran swap	Swap(State[i], State[j])	Swap(State[i], (State_Fact[j]mod 256))
	Penghitungan K Sequance	K = State[State[i]+State[j]] mod 256	K = State[(i+j) mod 256] +State[j+State[state[i]] mod 256]) mod 265